

COLLECTION FRAMEWORK ALL IN 1 :

21 February 2026 10:37

Collections in Java

Collections in Java play a crucial role in handling and manipulating groups of objects efficiently. In this blog, we will cover everything related to Collections, including why they are needed, their implementation, and a deep dive into the Collection framework.

What is a Collection in Java?

A Collection in Java is a framework that provides an architecture to store and manipulate a group of objects. It allows dynamic storage, retrieval, manipulation, and communication of object elements.

Why Do We Need Collections?

Before Collections, Java used arrays to store multiple elements. However, arrays had limitations:

- Fixed Size: Once declared, the size of an array cannot be changed.
 - No Built-in Methods: Arrays don't provide ready-made methods for searching, sorting, or modifying elements.
 - Inefficient Memory Management: Unused space is wasted, and resizing requires creating a new array.
- To overcome these issues, Collections were introduced, providing dynamic size and built-in utility methods.

Collection Framework Overview

The Java Collection Framework (JCF) is a hierarchy of interfaces and classes present in the `java.util` package. It provides efficient data structures for different types of collections.

Key Features of the Collection Framework:

- Reusable: Implements standard data structures like List, Set, and Map.
- High Performance: Optimized implementations for speed and memory.
- Extensible: Allows developers to create custom implementations.

Hierarchy of Collection Framework

Java Collections are divided into three main interfaces:

- List (Ordered collection, allows duplicates)
- Set (Unordered collection, no duplicates)
- Queue (FIFO, used for ordered processing)
- Map (Key-Value pairs, not part of Collection but an important data structure)

Java Collection Framework

Why Does Collection Extend Iterable?

`Collection` extends `Iterable<T>`, allowing it to be iterated using an iterator or an enhanced for-each loop.

Example:

```
import java.util.*;
public class CollectionExample {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");
        for (String fruit : list) {
            System.out.println(fruit);
        }
    }
}
```

Output:

```
Apple
Banana
Cherry
```

Methods of Collection Interface

The `Collection` interface provides several useful methods:

| Method | Description |
|--|---|
| <code>add(E e)</code> | Adds an element to the collection. |
| <code>addAll(Collection<? extends E> c)</code> | Adds all elements from another collection. |
| <code>remove(Object o)</code> | Removes a single instance of an element. |
| <code>clear()</code> | Removes all elements. |
| <code>contains(Object o)</code> | Checks if the collection contains a specific element. |
| <code>size()</code> | Returns the number of elements. |
| <code>isEmpty()</code> | Checks if the collection is empty. |
| <code>iterator()</code> | Returns an iterator for traversal. |

Example:

```
import java.util.*;
public class CollectionMethodsExample {
    public static void main(String[] args) {
        Collection<String> fruits = new ArrayList<>();
        fruits.add("Mango");
        fruits.add("Orange");
        fruits.add("Grapes");
        System.out.println("Collection: " + fruits);
        System.out.println("Contains Mango? " + fruits.contains("Mango"));
        System.out.println("Size: " + fruits.size());
        fruits.remove("Orange");
        System.out.println("After Removing Orange: " + fruits);
    }
}
```

Output:

```
Collection: [Mango, Orange, Grapes]
Contains Mango? true
Size: 3
After Removing Orange: [Mango, Grapes]
```

From <<https://www.codingshuttle.com/java-programming-handbook/collections-in-java/>>